Introducción a JavaScript

Historia

- Fue creado en 1995 por Brendan Eich en Netscape para hacer paginas web dinamicas.
- Aparece por primera vez en Netscape Navigator 2.0.
- JavaScript es una marca registrada de Oracle.
- Microsoft le dio el nombre de JScript.
- En 1997 para que JS sea adoptado como estándar propusieron a ECMA (European Computer Manufacturers Association) que haga el desarrollo.
- ECMA creó el comité TC39 para estandarizar el lenguaje. El primer estándar se llamó ECMA-262.
- En el estándar ECMA-262 se definió al lenguaje como ECMAScript.

Estándares ECMA I

VERSION	AÑO	Detalle
ECMAScript 1	1997	Primera versión.
ECMAScript 2	1998	No incluía nada nuevo sino que era un cambio de formato en la especificación para alinearse con los estándares ISO.
ECMAScript 3	1999	Añadió soporte para expresiones regulares, gestión estructurada de excepciones y algunas otras mejoras de menor calado, pero que convertían al lenguaje en algo más serio.
ECMAScript 4	----	Nunca salió a luz. Las luchas internas sobre la complejidad del lenguaje. Se querían añadir demasiadas cosas que hacían que el lenguaje perdiera parte de su propósito inicial.

Estándares ECMA II

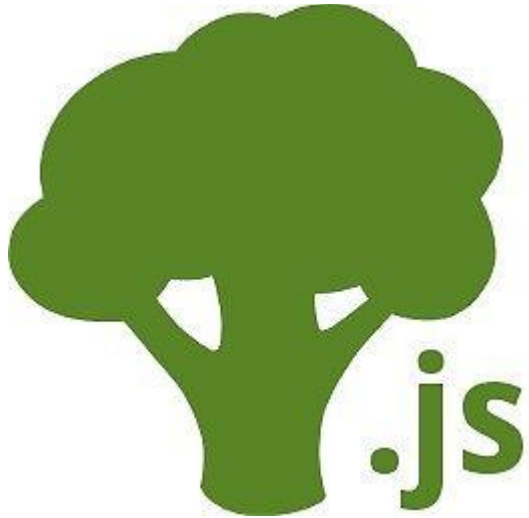
VERSION	AÑO	Detalle
ECMAScript 5	2009	Esta edición añadió el modo estricto del lenguaje, mejoró la especificación aclarando cosas que estaban demasiado abiertas en la versión 3 y que causaban confusión e implementaciones incoherentes, añadió el soporte nativo para JSON o los getters y setters para propiedades, entre otras pequeñas cosas.
ECMAScript 5.1	2011	Simplemente alineaba el estándar de ECMA con el formato correspondiente de ISO: ISO/IEC 16262:2011.
ECMAScript 6	2015	Lleva el lenguaje a un nivel superior, con muchas mejoras en cuanto a sintaxis, pero también conceptos nuevos como los símbolos o las lambdas, etc, etc...
ECMAScript 7	2016	En realidad lo que añade es mínimo: básicamente el operador de exponenciación y un método nuevo para las matrices.

ECMAScript en Navegadores

Navegador	Motor	Versión ECMAScript
Google Chrome	V8	6 Completo.
Firefox	SpiderMonkey	5.1 y mucho de 6.
Edge	Chakra	5.1 y mucho de 6.
Safari	JavaScript Core	5.1 y mucho de 6.
Internet Explorer	JScript 9.0	5.1

http://kangax.github.io/compat-table/es6/

Librerías de Compatibilidad y Transpiladores



Babel en acción

```
const nomVariable = n => n * n;
```

```
babel script.js --out-file script-compiled.js
```

```
var nomVariable = function nomVariable (n) {
  return n * n;
};
```

JavaScript

- Lenguaje Interpretado.
- Orientado a Objetos.
- Basado en Prototipos.
- Imperativo.
- Débilmente Tipado.
- Dinámico.
- Multi-paradigma

Tipos de Datos Primitivos

- Undefined.
- Null.
- Boolean.
- String.
- Symbol.
- Number.
  - NaN.
  - Infinity
    - POSITIVE_INFINITY.
    - NEGATIVE_INFINITY.
- Object.
  - Funciones.
  - Array.
  - Date.
  - RegExp.

Declaración de una Variable

```
var1 = new String("foo"); // crea un objeto String

var2 = String("bar"); //crea una Cadena primitiva

var3 = "foo"; // crea una Cadena primitiva

var3.length() // 3

var3.charAt(1) // o
```

Declaración de una Variable

var: Declara una variable.

let: Declara una variable de alcance local.

const: Esta declaración crea una constante que puede ser global o local a la función en la que se declara.

Declaración con var

- La sentencia var declara una variable, opcionalmente inicializándola con un valor.

```
var a = 'A';

var a = 0, b = 0;

function haz_algo() {

  var bar = 111;
  console.log(bar); // 111
}
```

Declaración con let

- La sentencia let declara una variable de alcance local.

```
if (x > y) {
  let gamma = 12.7 + y;
  i = gamma * x;
}
```

Declaración con const

- Las constantes presentan un ámbito de bloque tal y como lo hacen las variables definidas con let.

```
const MI_CONSTANTE = “Una constante”;
```

Hoisting

- Se conoce como hoisting a la acción que tiene el intérprete JS de mover todas las declaraciones al punto más alto de su scope.
- Las **asignaciones,** en cambio, se quedan donde están.

```
bla = 2;
```

```
var bla;
```

La variante Strict - Strict Mode

- ECMAScript reconoce la posibilidad de que algunos usuarios deseen restringir el uso de algunas de las características disponibles del lenguaje.
- Podrían hacerlo en interés de la seguridad, para evitar lo que consideran características propensas a errores, para obtener la comprobación de errores mejorada o por cualquier otra razón.
- El modo estricto y el uso de la sintaxis y semántica en modo estricto se hace explícitamente en el nivel de unidades de texto individuales de código ECMASCript.

```
"use strict";
var v1 = "Hola! Soy un modo estricto para script!";

v2 = 2;// esta línea lanza un ReferenceError
```

Objetos

- ECMAScript está basado en objetos.
- Un programa ECMAScript es un cluster de objetos comunicados.
- En ECMAScript un objeto es una colección de cero o más propiedades, cada una con atributos que determinan cómo se puede usar cada propiedad.
    - Las propiedades son contenedores que contienen a otros objetos, valores primitivos o funciones.
    - Un valor primitivo es uno de los siguientes tipos incorporados: Undefined, Null, Boolean, Number, String, y Symbol.
    - Un objeto es un miembro del tipo nativo Object.
    - Una función es un objeto invocable.
    - Una función que se asocia con un objeto a través de una propiedad se llama un método.

Declaración de Objetos

```
var o = {};

var o = new Object();

o.marca ='Ford';

o.modelo ='Ka';

var o = {

marca: 'Ford',
  modelo: 'Ka'

}
```

Atributos de una propiedad de datos I

Nombre Atributo	Dominio de Valores	Descripción
[[Value]]	Cualquier ECMAScript language type	El valor recuperado por un acceso get de la propiedad.
[[Writable]]	Boolean	Si es falso, los intentos de código de ECMAScript para cambiar atributo de la propiedad [[Valor]] utilizando [[Set]] no tendrá éxito.
[[Enumerable]]	Boolean	Si es verdad, la propiedad se enumeran por una enumeración for-in (ver 13.7.5). De lo contrario, la propiedad se dice que es no numerable.
[[Configurable]]	Boolean	Si es falso, los intentos de eliminar la propiedad, cambia la propiedad de ser una propiedad de acceso, o cambiar sus atributos (que no sean [[Valor]], o el cambio de [[Escribir]] false) fallará.

Atributos de una propiedad de datos II

Nombre Atributo	**Dominio de Valores**	**Descripción**
[[Get]]	Object | Undefined	Si el valor es un objeto debe ser un objeto de función. La función de [[CALL]] método interno (Tabla 6) se llama con una lista de argumentos vacía para recuperar el valor de la propiedad cada vez que una que se realiza el acceso de la propiedad.
[[Set]]	Object | Undefined	Si el valor es un objeto debe ser un objeto de función. La función de [[CALL]] método interno (Tabla 6) se llama con una lista de argumentos que contiene el valor asignado como su único argumento cada vez que un conjunto de acceso a la propiedad se lleva a cabo. El efecto de [[Set]] interna de una propiedad método puede, pero no está obligado, a tener un efecto sobre el valor que devuelve las llamadas posteriores al método interno [[Get]] de la propiedad.

Valores por Defecto de los Atributos

Nombre Atributo	**Valor por Defecto**
[[Value]]	undefined
[[Get]]	undefined
[[Set]]	undefined
[[Writable]]	false
[[Enumerable]]	false
[[Configurable]]	false

Algunos Métodos de Object

Funcion	Descripción
Object.create()	Crea un nuevo objeto con el prototipo específico y con propiedades.
Object.defineProperty()	Agrega a un objeto la propiedad nombrada descrita por un descriptor dado.
Object.defineProperties()	Agrega a un objeto las propiedades nombradas descritas por unos descriptores dados.
Object.keys()	Devuelve una colección que contiene los nombres de todas las propiedades enumerables propias de un objeto dado.
Object.getOwnPropertyNames()	Devuelve una colección que contiene los nombres de todas las propiedades enumerables y no-enumerables propias de un objeto dado.

Método defineProperty()

```
var o = {}; // Creates a new object

// Example of an object property added with defineProperty with a data property descriptor
Object.defineProperty(o, 'a', {
  value: 37,
  writable: true,
  enumerable: true,
  configurable: true
});

Object.defineProperty(o, 'b', {
  get: function() { return bValue; },
  set: function(newValue) { bValue = newValue; },
  enumerable: true,
  configurable: true
});
```

Método defineProperties()

```
Object.defineProperties(obj, {
  "property1": {
    value: true,
    writable: true
  },
  "property2": {
    value: "Hello",
    writable: false
  }
  // etc. etc.
});
```

Funciones

- Una función es un objeto invocable.
- En JavaScript, las funciones son **objetos de primera clase**, es decir, son objetos y se pueden manipular y transmitir al igual que cualquier otro objeto.
- En términos generales, una función es un "subprograma" que puede ser llamado por código externo (o interno en caso de recursión) a la función.
- Cada función en JavaScript es actualmente un objeto Function.
- Las funciones no son lo mismo que los procedimientos. Una función siempre devuelve un valor, pero un procedimiento, puede o no puede devolver un valor.

Declarando una Función con new

```
new Function ([arg1[, arg2[, ... argN]],] cuerpo-de-la-función)
```

```
var multiplicar = new Function("x", "y", "return x * y");

var laRespuesta = multiplicar(7, 6);
var miEdad = 50;
```

Declarando una Función con function

```
function sumar (num1, num2) {

    return num1 + num2;
}
```

Recibiendo los Argumentos

- El objeto arguments es una variable local disponible dentro de todas las funciones.
- Es posible invocar los argumentos de una función utilizando el objeto **arguments**.
- Una función puede tener hasta 255 argumentos.

```
function miConcat(separador) {
  var resultado = "";

  // Iterar a través de los otros argumentos enviados
  for (var i = 1; i < arguments.length; i++)
    resultado += arguments[i] + separator;

  return resultado;
}
```

Funciones Anónimas

```
var sumar = function (num1, num2) {


    return num1 + num2;
}

function saludador(nombre) {
  return function() {
    console.log("Hola "+nombre);
  }
}
var saluda = saludador("mundo");
saluda(); // Hola mundo
```

Funciones Flecha =>

```
var sumar = (num1, num2) => {

    return num1 + num2;
}

var cuadrado = num1 => {

    return num1 * num1;
}
```

```
var sumar =  (num1, num2) =>

    return num1 * num2;


var cuadrado = num1 => num1 * num1;
```

Funciones Autoejecutables

```
(function () {

    ........

})()
```

Creando un Objeto con new

```
function auto (num1, num2) {
    this.marca = "Ford";

    this.modelo = "Ka";
}

var unAuto = new  auto(num1, num2);

unAuto.marca = "Fiat";
```